Langara College
CPSC 2150
Instructor Gladys Monagan

Assignment #5: Sorting and Algorithm Efficiency

Assignment due with Brightspace at 11:50pm March 7

Reference and Learning Outcomes

Read in Drozdek chapters 2, chapter 9 (details under the *Week VI overview* in Brithspace

Read in Goodrich chapter 4 and 11.1, 11.2 and 11.3.

The purpose of this assignment is

- to design an experiment to determine the complexity of a function
- to make conclusions about the complexity of a function from empirical results
- to implement a sorting algorithm using a different data structure (other than an array or STL C++ vector)
- to find the empirical complexity of quicksort when sorting an array
- to find the empirical complexity of insertion sort when sorting a linked list
- to find the empirical complexity of mergesort when sorting a linked list
- to explain "system" limitations when running a program

Implement

You have a choice as to how you want to design and modularize this assignment. Document clearly in the document README.txt or README.pdf how your program should be compiled, linked, and run.

A) One main function that calls the various sorting algorithms, the timing functions, and outputs the results.
B) Three main programs, one for each of the sorting algorithms.

Time the various sorting algorithms to sort and determine the complexity of the best, worst, and average cases (based on input data) of

- quicksort when sorting **an array of integers** using the function included in this assignment, although you can optimize the code when there are very few elements left. Leave the function partition as is.
- mergesort when sorting **a linked list of integers**
- insertion sort when sorting **a linked list of integers**
  (you can use the code given in class if you so wish, just put "class notes" as a reference: up to you whether the implementation is recursive or iterative)

In order to do so, fill the array and fill the linked list with

a) random data
b) data in ascending order
c) data in descending order

For each of the three sorting algorithms with the required data structures,
there are 3 average timings (for data in random, ascending, and descending order) **per n** where n is the size of the array or of the linked list.

Present your timings as one or more tables and have your program do all the mathematical computations needed to show your conclusions in such a way that you can explain the complexity of quicksort, mergesort, and insertion sort.

Details of the Implementation

Just like you did for Assignment #4, to get an accurate average time to sort n elements, you need several trials.
Time how long it takes to sort an array (linked list) of n values m times and then divide the total time T by the number of trials m.

Remember not to sort already sorted arrays and not to sort already sorted linked lists between trials.

For all the sorting algorithms, start with n = 1000.
The number of trials m can (and probably will) vary from sorting algorithm to sorting algorithm and for the different data sets.

You can optimize to reduce the number of recursive calls when there are few elements left in the array. Also for the linked list (????).

Use the time Library <chrono> https://cplusplus.com/reference/chrono/
Look up how it is used and take the code on how to call it from samples online.
Remember that we are not concerned about the absolute times as such but the relative values compared to each other.
It seems that the high_resolution_clock from the chrono library is not implemented at Langara so, even if you use the high_resolution_clock for your results, please use the steady_clock when submitting your code so that we can compile and run your programs at Langara.

Conclusions

Include

- a README.txt or README.pdf with

1. the timings and results which are the output of your program(s): screenshots will work well if you can insert them into your document otherwise include the text output
2. for each of the 3 sorting methods using several values of n

   based on your timings and calculations of point 1.

   identify which is the best case, worst case, or average case of the algorithm.
   Determine the complexity: is it O(n), O(n logn) or $O(n^2)$?
   You can, for instance, do what we did in class where we divided the time function by n or n^2 … to look for a constant. Or the Ratio Test.
   Get the computer to do the mathematical operations and include them in your results. Do not do the computations by hand.

3. When does the best case occur for each of the sorting algorithms? And why?
   And the worst and average cases? Why?
   Discuss this.
   Explain your methodology of point 2 as to how you came up with the complexity based on your results.
   Bring in "system" limitations as observed:
   1. Possibly cost of calling new or delete
   2. Possibly cost of merging linked lists

To submit as Assignment #5 a single compressed zip file

As mentioned above, there could be one main program or three

- a program to run the quicksort tests
- a program to run the mergesort tests
- a program to run the insertion sort tests
- possibly a Makefile[1]
- a README.txt or README.pdf with your conclusions and timings

We should be able to run your code at Langara without having to fix any of your program(s).

The timings in your README document **do not** have to be the timings run on the Citrix machines at Langara. But your results should be reproduceable.

[1] If there are several files to be compiled and linked, provide a Makefile.
I do not recommend that you use C++ classes but if you do, submit a Makefile